Memory Leak (PART I)

AGENDA

I. Introduction

II. Dynamic Memory Allocation

III. Memory Management

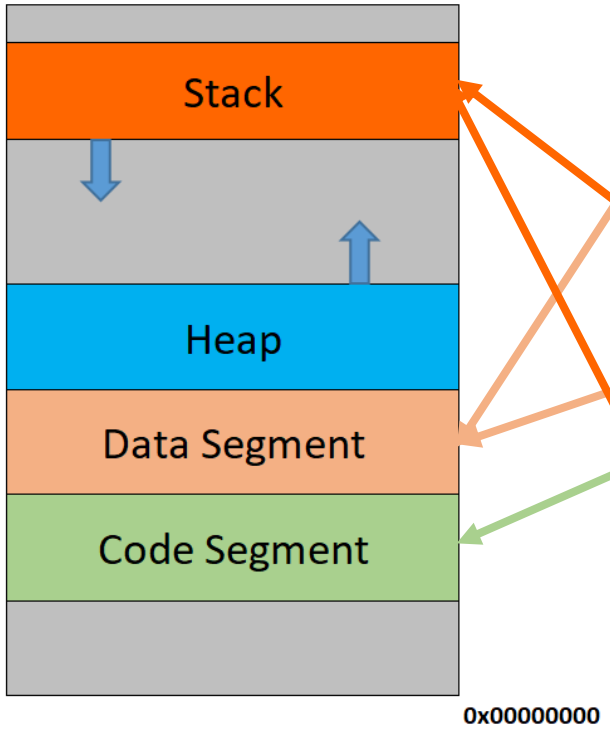
I - Introduction

Process memory

- When a process is started, virtual memory is allocated and is called address space
- Virtual address space provides an environment for the executable to run
- The size of data in a process depends on the platform
- Some parts of the process memory are used for storing data
  - global, static, local

I - Introduction

Process memory




```
int g_var1;                          // Uninitialized global variable
int g_var2 = 10;                     // Initialized global variable


void Incerement(int arg) {           // Ergument is a local variable
    int copy;                        // Another local variable
    copy = arg;
    static int counter = 0;          // Static variable
    counter += arg;
    std::cout << counter;
}

int main()
{
    float PI = 3.141f;               // Local variable
```

I - Introduction

Pointer

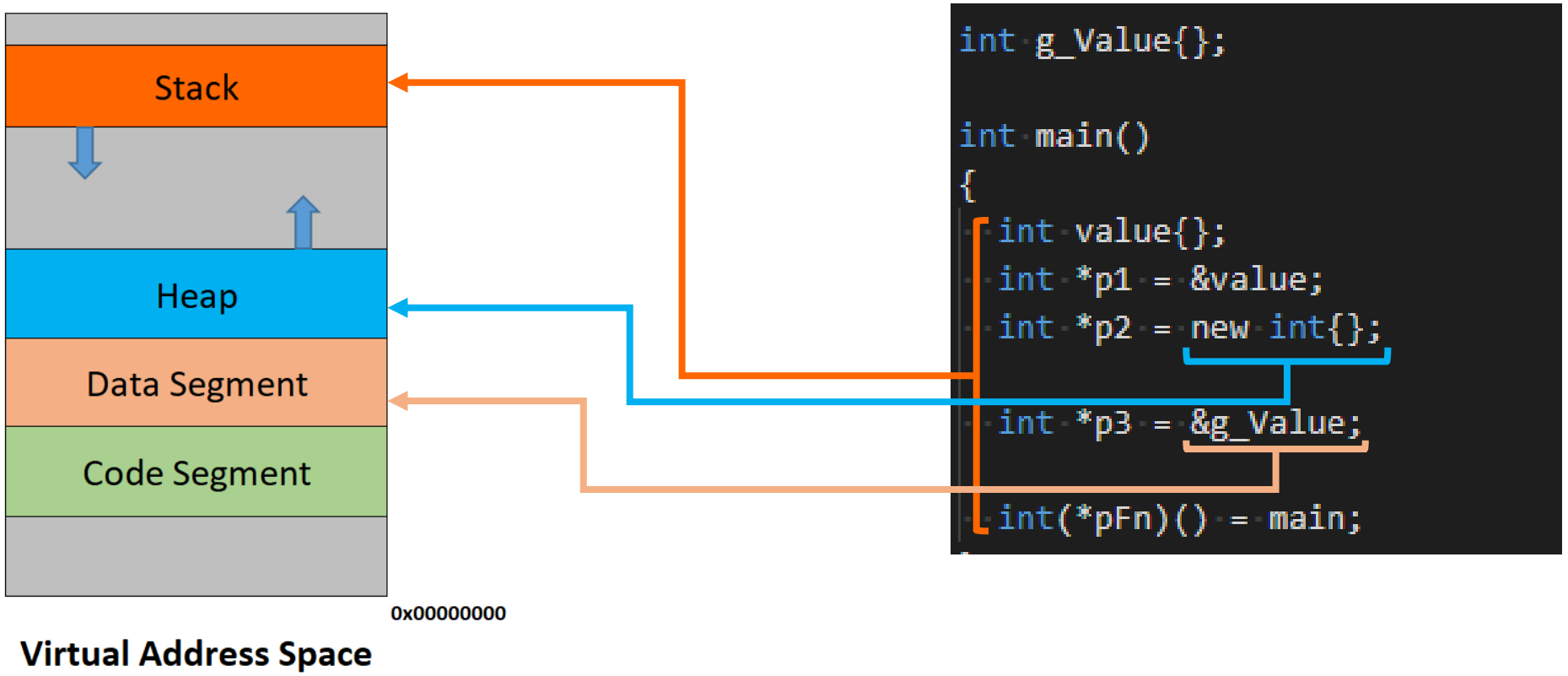
- A pointer is a variable, except that it holds address instead of the value
- Syntax is different from a variable

```
int *p = &<variable> of dynamic memory
```

- A pointer can hold address of:
  - Variable
  - Other Pointer
  - Object
  - Function

I - Introduction

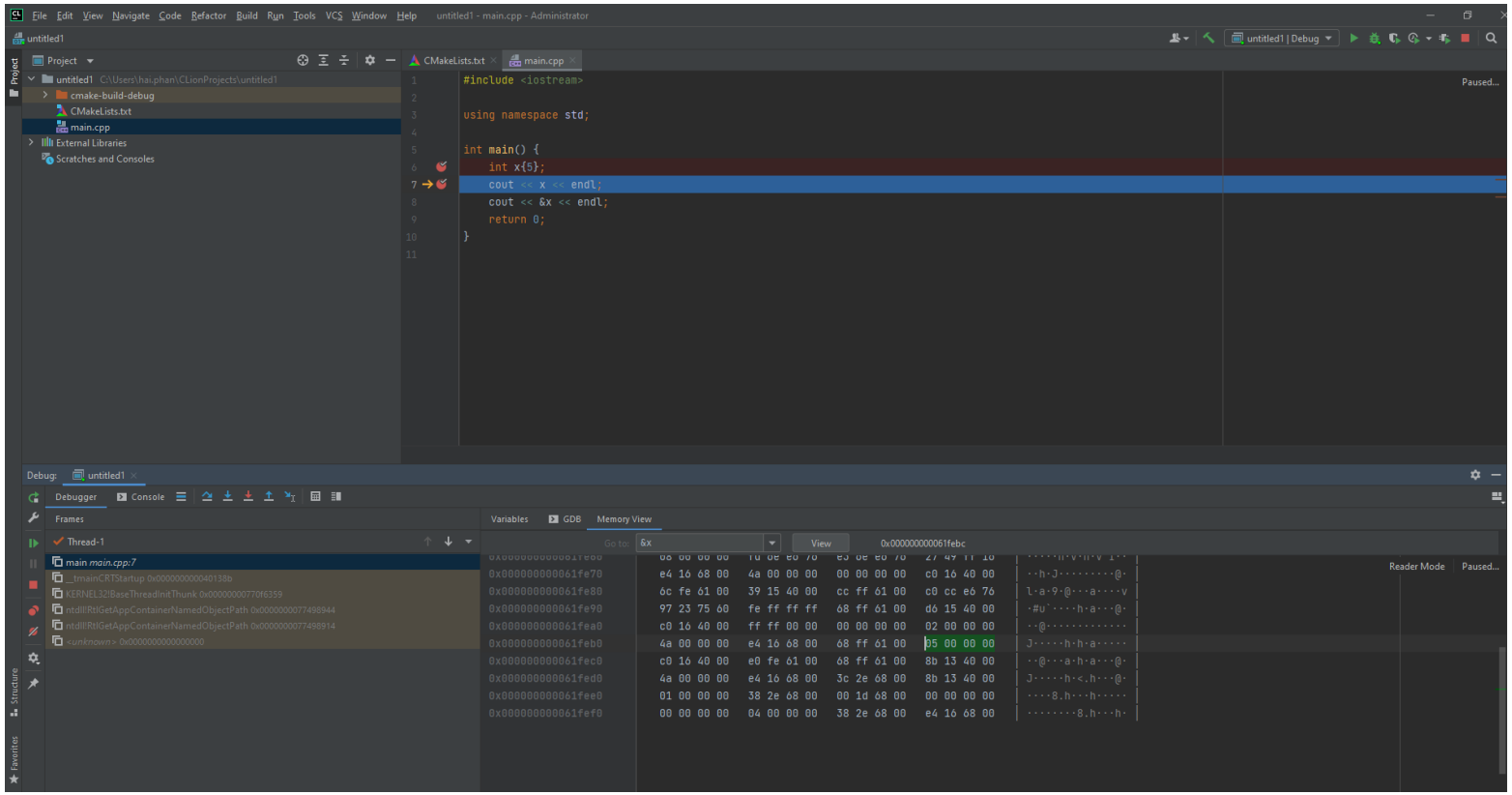
Pointer



I - Introduction

Development tool

- **CLion** uses the project model to inform its coding assistance, refactoring, coding style consistency, and other smart actions in the editor.



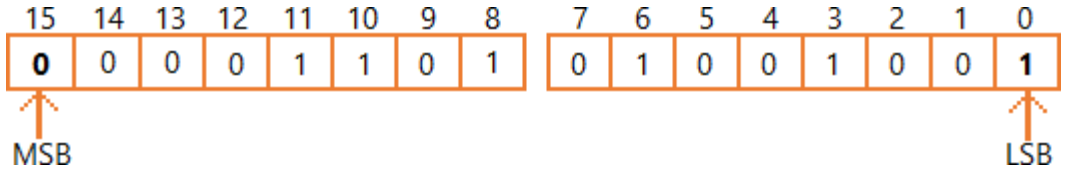
I - Introduction

Byte Ordering

- Specifies the order in which a sequence of bytes is stored in the memory
- Big Endian: most significant byte is stored first (the first byte in the data)
- Little Endian: least significant byte is store first (the last byte in the data)

I - Introduction

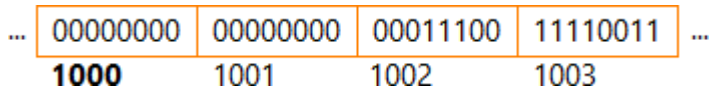
Byte Ordering



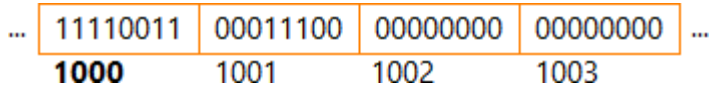
The number 7411 in binary:

1 1100 1111 0011

Big Endian



Little Endian



II - Dynamic Memory Allocation

Why Dynamic Memory:

- Required size of memory may be unknown at compile time
  - Loading information from external source
  - Build data structures with fixed size
- More control over lifetime of data
  - Lifetime of stack based data has limited by scope
  - Creating big objects

II - Dynamic Memory Allocation

Dynamic Memory Allocation in C:

- Refers to memory allocation on heap
- This memory is allocated at runtime
- Dynamic memory can be accessed throughout the program (as long as you have the address)
- Dynamic memory has to be released manually by the programmer.

II - Dynamic Memory Allocation

Dynamic Memory Allocation in C:

- C provides the following functions to allocate memory at runtime
  - malloc : allocates raw memory on the heap
  - calloc : allocates raw memory and initializes to zero
  - realloc : can change the size of memory block
  - free : releases the memory allocated by other functions

II - Dynamic Memory Allocation

Dynamic Memory Allocation in C:

- Not very useful in C++
- Cannot initialize memory
- Constructors cannot be invoked during allocation
- Destructor cannot be invoked during deallocation
- Cannot be customized
- Return NULL in failure

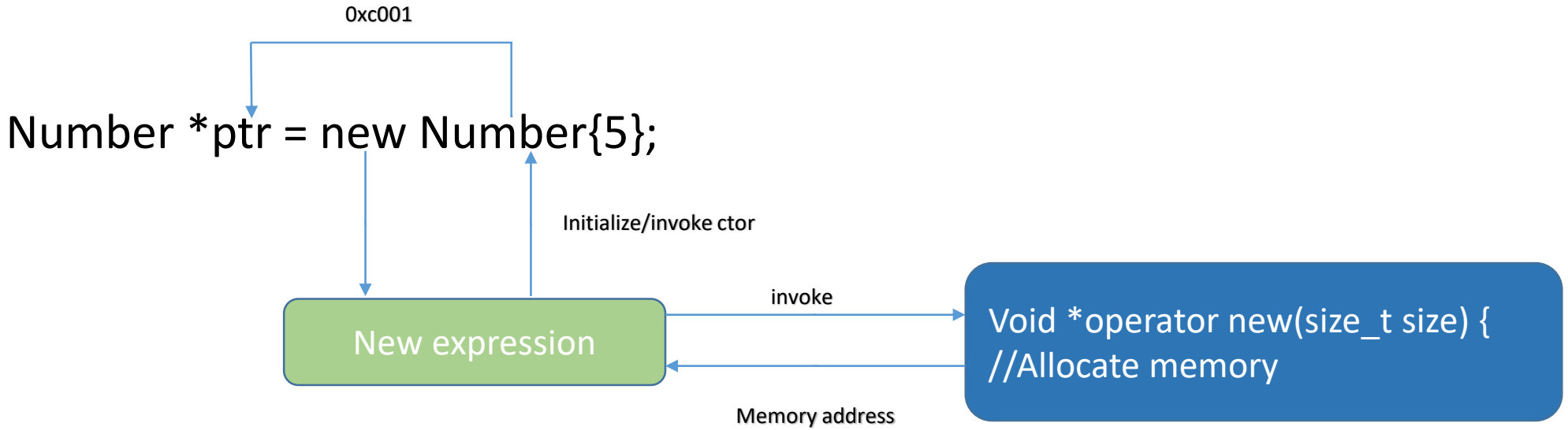
II - Dynamic Memory Allocation

Dynamic Memory Allocation in C++:

- C++ provides only two operators for allocating dynamic memory
- The operators have several forms:
  - New: allocates memory and optionally initializes it (by calling contructor)
  - Delete: releases memory after optionally calling a destructor
  - New[]: allocates memory for an array
  - Delete[]: releases memory for an array

II - Dynamic Memory Allocation (how new works):

- **Dynamic Memory Allocation in C++**



II - Dynamic Memory Allocation (how new works):

Handling new Failure - Exception:

- A new expression invokes operator new function
- Operator new function will try to allocate memory
- On failure:
  - It throws an exception of type bad_alloc (default behavior)
  - It calls a function set by set_new_handle function
  - It returns NULL (if nothrow version of new is used)

II - Dynamic Memory Allocation (how new works):

Non-throwing new:

- This new does not throw an exception if it fails, but returns nullptr
- It is useful when it may not be possible to catch the exception
- For example, you may be calling C++ code from C code. In such case, the exception if thrown, cannot be caught
- Or an application & library compiled with different compilers call each others' code that may throw exceptions. There is no guarantee that such exceptions can be caught by either application of the library

II - Dynamic Memory Allocation (how new works):

Non-throwing new (Syntax):

```
Object *pObj = new (std::nothrow) Object{};
//check the value of pObj for nullptr
```

II - Dynamic Memory Allocation (how new works)

Placement new :

- A new expression invokes operator new function
- Operator new function will try to allocate memory
- On failure
  - It throws an exception of type bad_alloc (default behavior)
  - It calls a function set by set_new_handler function
  - It return NULL (if nothrow version of new is used)
- On success:
  - The contractor is invoked
  - Memory address is returned

II - Dynamic Memory Allocation (how new works)

Placement new – Syntax:

```
Type *ptr = new (<memory address>) Type{};
```

II - Dynamic Memory Allocation (how new works)

Placement new :

- The form of new allocates memory at the address specified by the user
- Useful for creating object at designated memory locations
  - Useful in embedded devices where objects may be stores at known locations
  - Memory pooling
- You may need to call destructor explicitly for objects that have been created by placement new
- Standard library uses placement new in allocators

II - Dynamic Memory Allocation (how new works)

operator new:

https://en.cppreference.com/w/cpp/memory/new/operator_new

operator delete

https://en.cppreference.com/w/cpp/memory/new/operator_delete

malloc vs new

malloc	new
Function	Operator
Requires size during allocation	Size is ascertained from the type
Cannot initialize memory	Can initialize memory
Return void pointer that needs to be type casted	Returns correct type of pointer
Cannot be customized	Has different forms
Malloc, calloc & realloc	Has different forms
Return NULL or failure	Throws exception on failure

III - Memory Management

Memory Management Issues:

- Manual memory management can lead to various issues if not done correctly
- These issues arise due to various reasons
  - Failing to allocate memory for a pointer before using it
  - Allocating insufficient memory that leads to overwrite
  - Accessing memory after it has been freed
  - Freeing same memory twice
  - Not freeing memory after it is no longer required

III - Memory Management

Uninitialized Pointers

- A pointer holds a memory address
- If not initialized, it will contain an unknown value.
  - Possibly a valid memory address
  - Invalid memory address
  - Difficult to validate the pointer
- Accessing such memory address may lead to disastrous consequences
- Most compilers warn about uninitialized variables
- Always initialize pointers

III - Memory Management

Dangling Pointers

- After deleting memory, its address still remains in the pointer
- Consequently, the pointer continues pointing to the memory (which is no longer valid)
- Such a pointer is said to be dangling
- This can cause two problems
  - The same memory may be deleted again unknowingly (call double-delete)
  - Pointer may be used without knowing it’s dangling

III - Memory Management

Avoiding Dangling Pointers

- To avoid dangling pointer, assign *nullptr* immediately after deleting it
- If the pointer is accessed again, it will cause an access violation
- Deleting a null pointer is not an error; delete treats it like no-op

III - Memory Management

Memory leak

- Caused when memory is allocated, but not released afterwards
- The address may be lost and the memory can never be reclaimed
- If this happens with large allocation and/or repeatedly while the programming is running, the heap memory will run out
  - May cause the OS to increase the pagefile size
  - Will drastically reduce the performance of the system
  - Is worst case, your program will be killed by the OS

THANK YOU